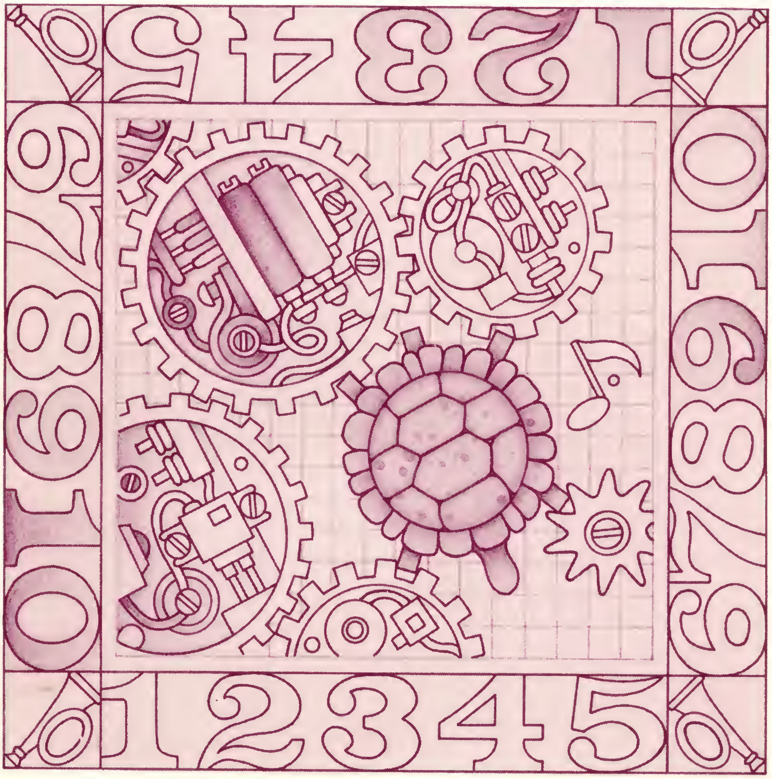
SIMULATED COMPUTER™

By: Jim Wieder
With: Scott Steketee

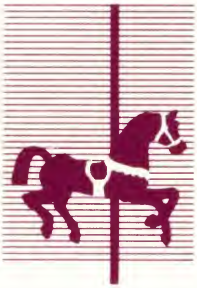
Learn how computers work. Sound and color graphics animate the inner workings of the computer as you type and run your own programs.



SIMULATED COMPUTER II

By: Jim Wieder
With: Scott Steketee

INTRODUCTION



Simulated Computer II demonstrates many of the principles of operation of a real computer. Like a real computer, it performs only one operation at a time, and works with numbers.

Simulated Computer II is easier to understand than a real computer because:

Simulated Computer	*Real Computer*
Uses ordinary decimal numbers (base 10)	Uses binary numbers (base 2)
Uses 24 memory locations	Uses many thousands of memory locations
Uses only 11 types of instructions	Uses 50 to 100 types of instructions (or more)
Executes 2 instructions per second (or less) so you can see what it's doing	Executes 100,000 instructions per second (or more)

Programming is giving the computer a list of instructions to perform (ie. execute). Learning to program Simulated Computer II can help you understand and visualize the inner workings of a real computer. The terms used in this simulation are very similar to most terms used by people who program in assembly language. Assembly language consists of instructions which are only three letters long. These shortened instructions are called mnemonics (nē mŏn' ĭks). ''Higher level'' languages, like BASIC, Pascal, and Logo are easier to use because their instructions are much closer to the words found in natural language.

Programming in assembly language requires great attention to detail. As you proceed through this instruction guide, there may be places you'd want to go over several times. You are also encouraged to depart from the text and experiment with programs of your own.

TABLE OF CONTENTS

LOADING INSTRUCTIONS

Be sure that you have removed all cartridges from your computer. A joystick is optional.

For Atari Computers:

Plug a joystick into ***port 1.***

If you are using a disk;

(1) Turn on your TV.
(2) Turn on the disk drive and wait for it to stop whirring.
(3) Put the disk into the drive and close the door.
(4) Turn on the computer. The program will be automatically loaded.

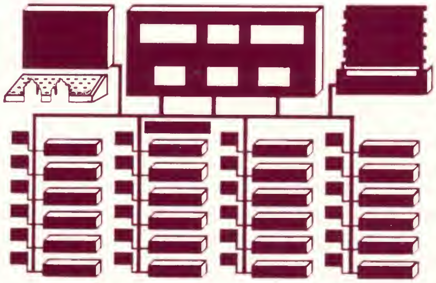
If you are using cassette;

(1) Turn on your TV.
(2) Insert the cassette into the tape machine. Rewind it if necessary.
(3) Press the PLAY button down on the tape machine.
(4) Press the START key on your computer (for 600XL and 800XL computers, also hold down the OPTION key) while you are turning on the power switch.
(5) After you hear the ''beep'' press the RETURN key.
(6) The program is now loading. This takes 9 minutes so please be patient.

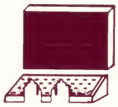
For the Commodore 64 computer:
Plug the joystick into port ***2.***
If you are using disk;
(1) Turn on your TV.
(2) Turn on the disk drive and wait for it to stop whirring.
(3) Turn on your computer.
(4) Put the disk into the disk drive and close the door.
(5) Type the following on your computer: LOAD''*'',8,1 (press RETURN) The program is loading into your computer.
(6) Type: RUN (Press RETURN.)

If you are using cassette;
(1) Turn on your TV.
(2) Turn on your computer.
(3) Place the cassette into the tape machine. Rewind it if necessary.
(4) Hold down the SHIFT key and press the RUN/STOP key at the same time.
(5) Press the PLAY button on the tape machine. The program is loading into your computer. This takes 12 minutes, so please be patient.

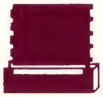
GETTING STARTED



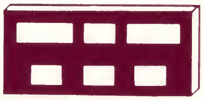
Look at the picture above. This is what you see when Simulated Computer II is ready to be used. The major parts of all computer systems are shown. They are explained below. Some of these parts can be seen just by looking at a computer. Others cannot be seen without taking the computer apart.



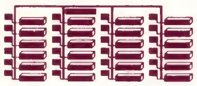
This represents an **input device** shown on the top-left corner of your TV screen. It is used to give information to the computer. This information may be numbers, instructions, or commands. In our case, the input device is the keyboard. When you type on the keys, the letters and numbers will be shown on this device.



At the top-right corner of your TV screen is an **output device.** It is pictured as a printer. The computer will give you information by printing it here. This may be such things as numbers or information which explains mistakes you've made.



This represents the **Central Processing Unit,** or **CPU.** Unlike the previous two parts, this is not plainly visible without taking the real computer apart and looking inside. The CPU is the center of action when programs are running. It controls the rest of the computer. Various pieces of the CPU are shown and will be described later when we use our first program.



These boxes, labeled 0 through 23, represent the **memory locations.** These are only visible by looking inside the real computer. The memory locations are where you will put information (instructions and numbers). Much of this booklet will concentrate upon showing you how to fill up these locations with instructions (a program) and numbers (data) to get the computer to do certain tasks.

LET'S DO SOMETHING

Type the word HELLO on your computer keyboard.
Did you notice that HELLO appeared on the input device while the simulated hands typed along with yours? Now press the RETURN key.
When you did, an ERROR message appeared on the output device. This INVALID COMMAND error means that the word HELLO is not in the simulated computer's language. (See Appendix IV for a complete explanation of ERROR messages.)
Now type HELLO again, but this time DON'T press the RETURN key. Instead, press the key which back spaces and erases each letter one at a time.

Atari computers:	DELETE BACK S
Commodore computers:	INST DEL

This is the way to erase typing mistakes, provided that you notice them before you press the RETURN key. Continue to erase the letters until the entire word, HELLO, has been erased.
To make a computer do something, you must load a program (ie. instructions) and other information into its memory. To start with a simple example, you'll program the simulated computer to add two numbers (6 and 4) and show the result on the output device. First you'll load these two numbers into memory locations 12 and 13. (We could've chosen any of the 24 memory locations.) You type:

LOAD12 (Press the RETURN key)

You must press the RETURN key after you finish typing each line. The following prompt is shown on the input device:

12:?

This means that the computer is ready to LOAD information into memory location 12. The first of our two numbers to be added is 6. You type:

6 (Press RETURN.)

Notice that the number 6 was sent to memory location 12, as instructed. 6 and 006 mean the same thing. The "leading zeros" are not necessary.

Now you can see that the computer prompt is:

13:?

Our second number to be added is 4, so you type:

004 (Press RETURN.)

Don't type the letter O when you mean the number zero. When RETURN was pressed, our data (ie. the number) was sent to memory location 13. We have no more data to load, so you type:

END (Press RETURN.)

END was not sent to memory location 14 because the computer understood END to mean, ''Stop LOADing information into memory''.

Now that the computer has some data, ie. 6 and 4, stored in its memory we'll LOAD the program which will add them together. Type the following: (Don't forget to press RETURN after you've finished each line.)

COMPUTER PROMPT	*YOU TYPE*	*THIS MEANS*
?	LOAD	LOAD the following instructions, beginning at memory location 00.
00:?	LDA12	Put a LDA12 instruction into location 00. Later, when we RUN this program, the computer will **L**oa**D** the contents of memory location **12** into the **A**ccumulator (AC). The accumulator is the place where all arithmetic is done. (If this is confusing to you, don't get discouraged. It will become more clear later when you RUN this program.)
01:?	ADD13	Put an ADD13 instruction into location 01. This instruction means **ADD** the contents of memory location **13** to the number that is in the accumulator.
02:?	STA12	Put a STA12 instruction into location 02. When executed, this instruction will **ST**ore the contents of the **A**ccumulator into memory location **12.**

03:?	OUT12	Put a OUT12 instruction into location 03. This instruction will **OUT**put the contents of memory location **12** to the output device.
04:?	STP	Put a STP instruction into location 04. This will **ST**o**P** the program from running.
05:?	END	I've finished LOADing my program.

What if I make a typing mistake?

If you make a mistake before pressing RETURN, use the BACK S or DEL key to erase it. If you don't notice that you've made a mistake until after you've pressed RETURN, type END (press RETURN). Then type LOAD followed by the number of the memory location which needs correcting and press RETURN. For example, if you just noticed that you made a mistake in location 02, do the following:

COMPUTER PROMPT	*YOU TYPE*	*THIS MEANS*
	END	Stop LOADing information into memory because I have to go back and correct something.
?	LOAD02	Get ready to **LOAD** the correct instruction into location **02.**
02:?	STA12	The correct instruction will now replace the incorrect one.
03:?	Continue on from here with the remainder of the instructions.	
	END	I've finished making corrections.

Now let's RUN the program to see if it works. You type:

RUN (Press RETURN.)

If you typed all of the above correctly, you should see a lot of action on the TV screen. Various locations on the screen light up and simulated electricity flows along the wires. Eventually, the program should show the number 10 on the output device. Ahah! The program added 6 and 4 together and showed us the result. What you saw was a simulation of what happens inside of a real computer when it does arithmetic.

You may have noticed that the first thing that happened after typing RUN was that all of the letters (ie. mnemonics) in the program instructions were changed to numbers. Computers must change assembly language instructions like these mnemonics to machine language, or number equivalents. They then can go ahead and execute the program. To get the mnemonics back again, type:

```
    ALP0-4      (Press RETURN.)
```

Be sure that you typed zero and not the letter O.
ALP stands for **ALP**habetic. This command means to change the numbers back to letters. The 0-4 in the above command tells the computer to change the instructions in locations 0 through 4.

The opposite of the ALP command is the DIG command. DIG stands for **DIG**its, or numbers. For example, try typing:

```
    DIG0-4      (Press RETURN.)
```

Notice that all of the mnemonics in locations 0 through 4 turn back into digits. At any time you choose, you can change from ALPs to DIGs to suit your needs. For example, you might use the ALP command after running a program, in order to help you see more clearly how to change or correct it.

Use the ALP command now to change the instructions in locations 0 through 4 back to mnemonics. RUN the program again and watch what happens. Type:

```
    RUN      (Press RETURN.)
```

Notice that this time the output printed on the output device is 14. What do you think the output would be if you were to RUN the program again? Try it and see!

LET'S TAKE A CLOSER LOOK

Most people find the sight of all of the activity of this computer simulation to be quite confusing at first. Let's get things back the way they were before you ran the program the first time. Then we'll describe the simulation in greater detail.

COMPUTER PROMPT	*YOU TYPE*	*THIS MEANS*
?	ALP0-4	Change the instructions in locations 0-4 to mnemonics.
?	LOAD12	Get ready to **LOAD** data beginning at location **12.**

12:?	006	Put a 6 into location 12. This is one of the original two numbers that we wanted to add.
13:?	END	I am done LOADing data.

We'll run the program once again. This time, however, we will make it run one step at a time. You type:

```
     RUNSTP      (Press RETURN.)
```

RUNSTP means **RUN** the program one **ST**e**P** at a time. Each succeeding step will be executed when you press the space bar, or push forward on the joystick. Read the explanations below as you step through the program. This part requires you to proceed slowly and to read carefully. If you get totally lost, you can stop the program by pressing the ''B'' key or the red button on the joystick. B stands for Break, which means to interrupt the program. Then you can start over by going back to the beginning of this ''Let's take a closer look'' section.

PRESS THE SPACE BAR. 00 appears in the **P**rogram **C**ounter (PC) to tell the computer where the first instruction is. 00 is placed in the FETCH box. This tells you that instruction 00 is about to be **FETCH**ed (ie. taken) and placed into the **I**nstruction **R**egister (IR).

PRESS THE SPACE BAR. 112 is FETCHed from location 00 and placed into the **I**nstruction **R**egister (IR). 112 is the numerical equivalent for the LDA12 instruction. ''PC'' appears in the INC box. This tells you that the **P**rogram **C**ounter is to be **INC**remented (ie. increased) by one. This is done so that the **P**rogram **C**ounter will be ''pointing to'' the next instruction to be fetched and executed.

PRESS THE SPACE BAR. PC, the **P**rogram **C**ounter, is changed to 01. LDA appears in the EXEC box. This means that the instruction about to be **EXEC**uted is LDA (ie. **L**oa**D** the **A**ccumulator).

PRESS THE SPACE BAR. Instruction 112 (LDA12) is executed. The leftmost digit of each instruction (1 in this case) is called the operation code and tells the computer what kind of instruction it is. 1, for example, is translated by the computer to mean **L**oa**D** the **A**ccumulator. The two right digits, 12 in this case, tell the computer where it is to find the number which we want to load into the accumulator. The computer takes a copy of the number that is in location 12 and puts it into the accumulator. If you are using our original data, this should be a 6. The **I**nstruction **R**egister (IR) contains the instruction being executed.

PRESS THE SPACE BAR. 313 is the next instruction to be executed. The digit 3, as explained above, is the operation code, or op code, for this instruction. The computer understands that this op code means ADD. The two digits 13, tell the computer where it is to find the number we want **ADD**ed to the number in the accumulator. After you **PRESS THE SPACE BAR TWO MORE TIMES,** you will see a 4 being sent to the accumulator and added to 6. This changes the value in the accumulator to 10.

PRESS THE SPACE BAR. 212 is the next instruction. The op code here is 2, which the computer understands to mean **ST**ore the contents of the **A**ccumulator (STA). The 12 tells the computer where it is to store the number that is in the accumulator. In this case, the accumulator contains the number 10, which it stores into location 12. **PRESS THE SPACE BAR TWO MORE TIMES** and you will see this instruction executed. Notice that the 6, which used to be in location 12, has been destroyed.

PRESS THE SPACE BAR. 812 is the next instruction. 8 is the op code for an **OUT**put instruction. The computer translates 812 to mean **OUT**put the contents of location **12** to the output device. **PRESS THE SPACE BAR TWO MORE TIMES** and you will see a 10 (the contents of location 12) being sent to the output device.

PRESS THE SPACE BAR. The next instruction, 000, is translated by the computer to mean **ST**o**P. PRESS THE SPACE BAR THREE MORE TIMES** and the **ST**o**P** instruction will be executed. When the program is done, a prompt (question mark) will appear on the input device to tell you that the computer is ready for more commands.

All of this may seem like a lot of effort for just a simple addition problem. Real computers work in a similar fashion to the steps described here, but they do this work extremely fast. What took our simulated computer about half a minute to do, would take a real computer a tiny fraction of a second.

ANOTHER INSTRUCTION

If you are just turning on your computer, you'll have to LOAD the program and data from page 7 before continuing. If your instructions are all digits, change them to mnemonics by using the ALP0-4 command.

Now change the instruction in memory location 4 from STP to JMP00. JMP00 means to **J**u**MP** to location 00. When this instruction is executed, the computer will change the program counter (PC) so that it "points to" the instruction in location 00. Here's how you LOAD this instruction into the program.

COMPUTER PROMPT	*YOU TYPE*	*THIS MEANS*
?	LOAD04	Get ready to **LOAD** into location **04.**
04:?	JMP00	Put a JMP00 instruction into location 04.
05:?	END	I am done LOADing information.

What do you think JMP00 will make the program do? RUN it to see. Type:

RUN (Press RETURN.)

Watch the program for a while and see if you can predict what it will do next. To stop the program, press the B (for **B**reak) key or hold down the red button on the joystick. To continue the program type:

CONT (Press RETURN.)

(Note: It is possible to stop the program at the precise instant when there is a 5 in the program counter. In this case, when you try to **CONT**inue, the computer will look into location 5 and find no instruction. It will then show you an ERROR-G message. You have no alternative but to type RUN again.)

Did you see that the program will seemingly never end because it is forever "looping" through the program instructions? Programmers call this an "infinite loop". In this case, however, the program would stop eventually because the numbers will get too large for the simulated computer.

Let's see just how high the simulated computer can count. To test this limit more quickly than the previous program can do, we will make two changes to our data in locations 12 and 13. If you haven't already done so, stop the above program by pressing the B key. Then do the following:

COMPUTER PROMPT	*YOU TYPE*	*THIS MEANS*
?	LOAD12	Get ready to **LOAD** some data into location **12.**
12:?	600	Put 600 into location 12.
13:?	100	Put 100 into location 13.
14:?	END	I'm done LOADing data.

Write down the first few numbers that you think the computer will output. Then RUN the program to see if you were right. Let it keep running until it stops by itself.

How did the computer tell you that it could not execute the next step? By printing an ERROR-A message on the output device. ERROR-A is the computer's way of telling you that it couldn't execute an instruction because the result of a calculation would be too large or too small.

Notice that the number 900 was in the accumulator when it stopped running. The program was trying to add 100 to the accumulator. From this we can tell that the largest number the simulated computer can work with is somewhere between 900 and 1000.

Here's a challenge for you. Find the largest number by changing the data in locations 12 and 13. (See Appendix I for a sample solution.)

Next, try to find the smallest number that the simulated computer can work with. (Hint: Try using the SUB instruction described on page 33. A sample solution is found in Appendix I. Another way is to use the above program but have it add negative numbers to the total.)

COMPUTER SOUND

Type: **NEW** (Press RETURN.)

NEW is a command which clears all of the computer's memory and prepares it for completely **NEW** information. When you first turn on your computer, memory is automatically cleared.

Perhaps you noticed the letters S, C, T, and D to the right of memory locations 20, 21, 22, and 23. These memory locations have special jobs. Location 20, labelled S, is used for sound. When a STA20 instruction is executed, a sound is produced. Let's explore this sound generator by typing in the following:

COMPUTER PROMPT	*YOU TYPE*	*THIS MEANS*
?	LOAD	Get ready to **LOAD** some information beginnig at location **00.**
00:?	LDA07	Put a LDA07 instruction into location 00. When this instruction is executed, the computer will **L**oa**D** the **A**ccumulator with the number that it finds in location **07.**
01:?	STA20	Put a STA20 instruction into location 01. When this instruction is executed, the computer **ST**ores the number that is in the **A**ccumulator into location **20.** **Whenever a number is stored into this special location, a sound is produced.**
02:?	JMP01	Put a JMP01 instruction into location 02. When executed, the computer **J**u**MP**s to the instruction in location **01.**
03:?	press RETURN	Pressing the RETURN key advances you to the next memory location.
04:?	press RETURN	
05:?	press RETURN	
06:?	press RETURN	
07:?	30	Put the number 30 into location 07.
08:?	END	I am done loading information.
?	RUN	RUN the program.

Watch and listen to what happens when this program is RUN. You may need to adjust the volume on your TV. This program makes an occasional "beep" sound.

You can see that the computer must take many actions for each program instruction. Because the simulated computer takes the time to show you all of these steps, it would not be possible to produce recognizable tunes with it. It is possible, however, to run programs at somewhat faster or slower speeds. Press the B (Break) key to stop the above program from running. Now type the following:

RUNSPED9 (Press RETURN.)

RUNSPED9 means to **RUN** the program at **SPE**e**D 9.** The number can range from 0 (the slowest) to 9 (the fastest). When you use the ordinary RUN command, the speed is automatically set to 4.

If you have a joystick, you can also use it to control the speed with which a program runs. Press the B key and type the following:

RUN (Press RETURN.)

This, of course, is an ordinary RUN command. Now push forward on your joystick and you will notice that the run speed will increase. Pulling backward on the joystick will decrease the speed of running. The output device will display a message that tells you how fast the program is running.

Now press the B key to stop the program. Load a different number into location 07 and RUN the program again. Did you hear a sound which is higher or lower? Experiment with several numbers in location 07. You will find that if your program tries to store a number less than 1 or greater than 37 (Atari computers) or 70 (Commodore 64) into location 20, it stops and displays an ERROR-I message on the output device.

As you watched this program run, you can see that the simulated computer goes through the same process as it did when it added two numbers together in our first program example. That is, it:

(1) converts all of the mnemonics to numbers;
(2) sets the program counter to the first instruction;
(3) FETCHes the instruction from the memory location "pointed to" by the **P**rogram **C**ounter (PC); and places the instruction into the **I**nstruction **R**egister (IR);
(4) **INC**rements the **P**rogram **C**ounter (PC);
(5) **EXEC**utes the instruction which is contained in the **I**nstruction **R**egister (IR).
(6) Each additional instruction is executed by repeating steps 3, 4, and 5 above.

Now we will write a program which will make a series of different sounds. This will be done in similar fashion to the "counting" program done earlier. We will also show you that it is possible to load the data beginning at location 00 to be followed by the program instructions. Type in the following:

COMPUTER PROMPT	*YOU TYPE*	*THIS MEANS*
?	NEW	Clear the computer's memory and prepare for **NEW** information.
?	LOAD	Get ready to LOAD.
00:?	0	Put the number 0 into location 0.
01:?	5	Put the number 5 into location 01.
02:?	LDA00	Put a LDA00 instruction into location 02. Recall that this will **L**oa**D** the **A**ccumulator from location **00** (when executed).
03:?	ADD01	Put a ADD01 instruction into location 03. When executed, this will **ADD** the contents of location **01** to the accumulator.
04:?	STA20	Put a STA20 instruction into location 04. When executed, this will **ST**ore the contents of the **A**ccumulator into location **20.** That is the special location that we use to make sounds.
05:?	JMP03	Put a JMP03 instruction into location 05. This will cause the program to **J**u**MP** to the instruction that is in location **03.**
06:?	END	I'm done LOADing.
?	RUN02	(Don't forget the 02!) **RUN** beginning at location **02.**

Listen carefully and you will hear that the beeps gradually change in pitch. You can change the speed of this program with your joystick. When you are through watching the program run, press the B key to stop it.

TURTLE GRAPHICS

By using the special memory locations labelled **C, T,** and **D** (21, 22, and 23), you can write programs which will draw pictures on your TV. Simulated Computer II uses a technique called ''turtle graphics'' to accomplish this. The turtle, as it has come to be known, can be instructed to turn and draw lines of various lengths and colors. This is best shown by an example. Try this:

COMPUTER PROMPT	*YOU TYPE*	*THIS MEANS*
?	NEW	Clear the computer's memory.
?	LOAD	Get ready to **LOAD** information beginning at location **00.**
00:?	LDA19	When executed, this instruction will **L**oa**D** the **A**ccumulator from location **19.**
01:?	STA23	When executed, this instruction **ST**ores the **A**ccumulator to location **23.** Similar to the way that executing a STA20 causes a sound to be made, a STA23 causes the turtle to draw a line. The length of the line is determined by the number that is stored there.
02:?	JMP01	Put a JMP01 instruction into location 02.
03:?	END	I'm done loading my instructions.
?	LOAD19	Get ready to **LOAD** some data, beginning at location **19.**
19:?	10	Put the number 10 into location 19. This will be the length of the line that the turtle draws.
20:?	press RETURN	
21:?	2	Put the number 2 into special location 21. This location (labelled C) is special because the number it con-

		tains determines the **C**olor that the turtle will draw. Look at page 24 to find out the meaning of the color values.
22:?	90	Put the number 90 into special location 22. This location is labelled T which stands for **T**urn. The value in this location determines the number of degrees that the turtle turns before it draws a line. The turtle rounds off this number to the nearest 15 degrees. In other words, the only turns that the turtle can actually make are 15, 30, 45, 60, 75, 90, 105, etc. degrees.
23:?	END	I'm done LOADing.
?	RUN	RUN the program.

Watch what happens. Eventually, the turtle draws a square. Each time a STA23 instruction is executed, the turtle screen is shown. The turtle turns the number of degrees which is contained in location 22, and draws with the color value found in location 21. Colors may vary depending upon the color settings on your TV, or upon the direction the turtle is pointing. Press the B key to stop the program. The turtle screen will now be shown. Pressing any key (EXCEPT THE RESET KEY!) will return you to the simulated computer display. If you should want to see the latest turtle screen at any future time, just press the OPTION or F7 key found on the right side of your keyboard. Again, pressing any other key (except RESET) will return you to the simulated computer display. If you want to get the program running again from where it left off, type CONT and press RETURN.

Let's expand upon our square-making program by adding instructions which keep track of the number of times the turtle has drawn a line. In this way we will be able to tell it to stop after it has drawn the fourth side of the square. We will use two other locations for data. Location 17 will contain the value which "counts down" the number of times the turtle has drawn. Location 18 will contain the number 1. This is the amount which is subtracted from the "counter" in location 17 after each line is drawn. If we start with 4 in location 17, the program will be told to stop when the number in that location has counted down to 0. Here we go.

COMPUTER PROMPT	*YOU TYPE*	*THIS MEANS*
?	ALP0-2	Change back to mnemonics.
?	LOAD02	Get ready to **LOAD** at location **02.**
02:?	LDA17	**L**oa**D** the "counter" into the **A**ccumulator.
03:?	SUB18	**SUB**tract the value in location **18.**
04:?	STA17	**ST**ore the new counter from the **A**ccumulator into location **17.**
05:?	SKP03	**SK**i**P** over the next instruction if the value in the accumulator is 0. This instruction is the one which decides when our turtle stops. (There are other kinds of **SK**i**P** instructions described in Appendix II.)
06:?	JMP00	**J**u**MP** back to the beginning.
07:?	END	I'm done LOADing my program.
?	LOAD17	Get ready to LOAD some data.
17:?	4	the counter
18:?	1	the number to be subtracted from the counter
19:?	15	a change of length
20:?	press RETURN	
21:?	12	a change of color
22:?	90	the number of degrees the turtle turns
23:?	END	I'm done loading.
?	RUN	RUN the program.

Watch as the program runs for a while. After it draws the fourth side of the square it does stop, but our triumph is discolored a bit by an ERROR-G message on the output device. The computer was trying to get an instruction from location 7. When it found nothing there, it gave us the ERROR-G message. Load a STP instruction into location 7. Then Load the original counter (4) back where it belongs and RUN the program again. It should then stop without giving the error message.

We'll now expand upon our program even farther by adding instructions which will allow us to make many squares of various sizes. Two things must happen before our program goes on to do each succeeding square. First the counter must be reset to 4. Second, the computer must be told what size square to make. Both of these can be done with INP (**INP**ut) instructions. When the computer executes an INP instruction, it will stop running and wait until you type a number and press RETURN. That number is then sent to the location you have specified in the INP instruction. For example, an INP17 instruction will send the inputted number to location 17. We'll change our existing square drawing program by adding two INP instructions. So far, our program looks like this:
(You may need the ALP command to see this.)

```
    00:  LDA19
    01:  STA23
    02:  LDA17
    03:  SUB18
    04:  STA17
    05:  SKP03
    06:  JMP00
    07:  STP
```

Now we will change this program as follows:

COMPUTER PROMPT	*YOU TYPE*	*THIS MEANS*
?	LOAD07	Get ready to **LOAD** at location **7.**
07:?	INP17	Put a INP17 instruction into location 7. When executed, this will cause the program to stop and wait for the person at the keyboard to type a number. When he presses RETURN, that number will be sent to location 17. In our program, this is the way we reset the counter to 4.

08:?	INP19	When executed, this instruction will cause the program to wait for the person at the keyboard to type a number. When RETURN is pressed, it sends that number to location 19. That is the place where the program will find out how big to draw the square.
09:?	JMP00	**J**u**MP** back to the beginning.
10:?	END	I'm done LOADing instructions.
?	LOAD17	Get ready to LOAD data.
17:?	4	Put back the original counter.
18:?	END	I'm done LOADing. (You should still have some data in locations 18, 19, 21, and 22.)
?	RUN	RUN the program.

After this program draws a square with 15 units per side, it will prompt you with a number sign (#) and wait for you to type the number 4 and press RETURN. (Do you remember that this resets the counter?) After it sends the 4 to location 17, the computer prompts you again. It's waiting for you to type a number which it will send to location 19. Type a number (not too big) and press RETURN. Did the program use your number to draw a different sized square? This program will continue in this fashion as long as you are willing to give it numbers to use. Continue experimenting with a few more numbers. When you are done, input a B instead of a number. This will stop the program.

Why not add another INP instruction to this program to allow the user to change the color of the square too? In fact, why stop there when you can use an INP instruction to change the size of the turn too.

CHALLENGE: Write a program which initializes the counter to 4 without using an INP instruction.

CHALLENGE: Write a program which can draw a spiral.

(See sample solutions in Appendix I.)

SUMMARY OF SOUND AND GRAPHICS IN SIMULATED COMPUTER II

LOCATION	*LABEL*	*HOW IT IS USED*
20	S	Sound: When a STA20 instruction is executed, a sound is made. The number stored here determines the pitch.
21	C	Color: This location contains a number which determines the color which the turtle uses to draw.
22	T	Turn: This location contains a number which tells the turtle how many degrees (rounded to the nearest 15) to turn before drawing.
23	D	Draw: When a STA23 instruction is executed, the turtle turns and draws a line. The length of the line is determined by the number which is stored here.

The turtle always starts pointing ''north'' (straight up). Turns of positive degrees are clockwise.

Colors can be changed while a program is running, but only 3 different colors can appear on the screen at the same time.

TABLE OF COLOR VALUES:

When the following numbers are in the color location 21, they determine the color of the line that the turtle draws.

For Atari computers:

Value	*Color*
2	gray
16	gold
32	orange
48	red-orange
64	pink
80	purple
96	blue-purple
112	blue
128	blue
144	light blue
160	turquoise
176	blue-green
192	green
208	yellow-green
224	orange-green
240	light orange

The even numbers between each color value increase the brightness of that color. For example, 32, 34, 36, 38, 40, 42, 44, and 46 are all shades of orange, from dark to very bright.

for Commodore 64 computers:

Value	*Color*
0	black
1	white
2	red
3	cyan
4	purple
5	green
6	blue
7	yellow
8	orange
9	brown
10	light red
11	dark gray
12	medium gray
13	light green
14	light blue
15	light gray

A SPECIAL INPUT INSTRUCTION

When using an INP instruction, it is frequently necessary to follow it up with a LDA instruction so that something useful can be done with the number you input. For this reason, there is a special INP instruction which combines these two instructions. It is INP24. Notice that there is no memory location 24. For the purposes of this instruction, that location is taken to be the accumulator. Following are two programs, both of which do the same thing (make a sound). You can see that the second one uses one less instruction and therefore less time and memory.

```
00:  INP12           00:  INP24
01:  LDA12           01:  STA20
02:  STA20           02:  STP
03:  STP
```

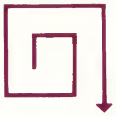
A SPECIAL OUTPUT INSTRUCTION

Just as an INP24 instruction causes the program to INPut your number directly into the accumulator, an OUT24 instruction will OUTput the number that is in the accumulator. This eliminates the necessity of first storing the number into a memory location so less time and memory are used.

A COLLECTION OF CHALLENGES

We can learn a great deal from examples. You have been shown all of the major "building blocks" (commands and instructions) of Simulated Computer. Appendix II and III list them all in one place with brief explanations. You may notice that there are several kinds of SKP instructions which we have not used thus far. We have also not used the MUL and DIV instructions. Appendix IV contains a list of the error messages which you may encounter. After you review these appendices, challenge yourself with the following exercises. Compare your solutions with those suggested in Appendix I.

The following challenges are presented to give support to your explorations. It is our hope that you will use them as "springboards" for your own designs and challenges.



CHALLENGE 1: THE SQUIRAL

Write a program which draws a square spiral, or "squiral". This program should use no INP instructions.

CHALLENGE 2: EVALUATE ALGEBRAIC EXPRESSIONS

Those of you who have studied algebra may recall that you were occasionally required to calculate the value of an expression when particular values are substituted for variables. For example, the expression 7X is equal to 14 if X is 2. The value is 42 if X is 6. The value of the expression $3X-5$ is 13 if you substitute 6 for X. (Note: 3X means 3 times X.) Write a computer program which will evaluate each of the following expressions. Check your programs by running them and inputting the numbers given below.

Expression	Inputs	Computer should output
(a) $X-5$	7	2
	5	0
	-77	-82
(b) $7X-8$	3	13
	43	293
(c) $3A+5B$	$A=2, B=5$	31
	$A=-5,$ $B=3$	0
(d) X^2+2X-1	3	14
	0	-1
	-3	2

CHALLENGE 3: NUMBER SEQUENCES

A number sequence is a series of numbers that follow a particular "rule", or pattern, as they go from one to the next. For example, the sequence:

(a) 3, 6, 12, 24, 48, 96, etc.

uses the rule "multiply by 2". Can you guess the rule for the following sequence?

(b) 2, 5, 11, 23, 47, 95, etc.
(It is "times 2 then add 1".)

Once you know the rule for a sequence, you can write a program which will output that sequence. Write programs which output the above sequences. Then try your hand at the following sequences:

(c) 2, 5, 14, 41, 122, etc.

(d) The following sequence is quite famous. It appears in many forms in nature. It is called the Fibonacci sequence. It's rule is somewhat different than the above examples.

1, 1, 2, 3, 5, 8, 13, 21, 34, etc.

CHALLENGE 4: A DECISION MAKER

(a) Write a program that uses an INP instruction. If the number that you INPut is positive or zero, have your program OUTput the number 1. If the number that you INPut is negative, have the program OUTput the number zero.

(b) This exercise is an extension of the above. If the user INPuts a positive number or zero, have your turtle turn right and draw a line 10 units long. If a negative number is INPut, the turtle must turn left and draw the line.

(c) Write a program which determines whether the value the user INPuts is less than or equal to 7. OUTput a 1 if it is, and a 0 if it is not.

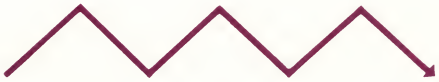
(d) In exercise 1 you were asked to write computer programs which evaluate expressions like $X-5$ or $7X$. Now write a program which can tell you if:

$$x^2 - 3 = 2X$$

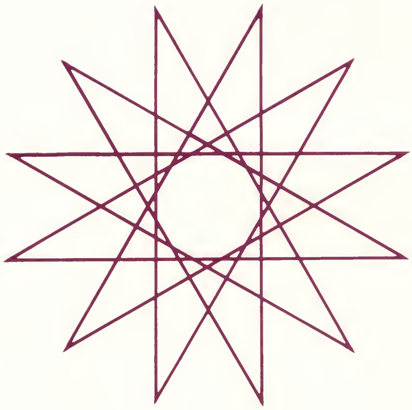
If it is true for the user's INPut, then OUTput a 1. Output a 0 if it is not true. Run the program for the following INPuts:

$X=2$	not true
$X=3$	true

CHALLENGE 5: A SAWTOOTH
Write a program which draws a ''sawtooth'' line like the one shown.



CHALLENGE 6: A 12-POINTED STAR
Did you discover that the turtle can draw backwards if you store a negative number to the Draw location? Use this fact to draw a 12-pointed star like the one shown.



APPENDIX I : *SAMPLE SOLUTIONS TO THE CHALLENGES*

from page 14: Write a program to find the largest number that Simulated Computer II can work with.

```
LOAD          LOAD12
00: LDA12     12: 995
01: ADD13     13: 1
02: STA12     14: END
03: OUT12
04: JMP00
05: END
```

from page 14: Write a program to find the smallest number that Simulated Computer II can work with.

```
LOAD          LOAD12
00: LDA12     12: -995
01: SUB13     13: 1
02: STA12     14: END
03: OUT
04: JMP00
05: END
```

from page 22: Write a program which initializes the counter to 4 without using an INP instruction.

```
LOAD          LOAD21
00: LDA11     21: 12        (color)
01: STA13     22: 90        (turn)
02: INP19     23: END
03: LDA19
04: STA23
05: LDA13
06: SUB12
07: STA13
08: SKP03
09: JMP03
10: JMP00
11: 4
12: 1
13: END
```

from page 22: Write a program which can draw a spiral.

```
LOAD          LOAD21
00: INP12     21: 12        (color)
01: LDA12     22: 45        (turn)
02: STA23     23: END
03: JMP00
04: END
```

When this program s RUN, you will be prompted to INPut the draw numbers. For example, INPuting the numbers 2, 4, 6, 8, ...24 will produce a spiral.

CHALLENGE 1: SQUIRAL

```
LOAD          LOAD21
00: LDA04     21: 12        (color)
01: ADD05     22: 90        (turn)
02: STA23     23: END
03: JMP01
04: 0
05: 2
06: END
```

CHALLENGE 2: EVALUATING EXPRESSIONS

(a) $X - 5$

```
LOAD
00: INP24    (X)
01: SUB04
02: OUT24
03: JMP00
04: 5
05: END
```

(b) $7X - 8$

```
LOAD
00: INP24    (X)
01: MUL05
02: SUB06
03: OUT24
04: JMP00
05: 7
06: 8
07: END
```

(c) $3A + 5B$

```
LOAD
00: INP18    (A)
01: INP19    (B)
02: LDA18    (A)
03: MUL10    (3A)
04: STA18
05: LDA19    (B)
06: MUL11    (5B)
07: ADD18    (3A + 5B)
08: OUT24
09: JMP00
10: 3
11: 5
12: END
```

(d) $X^2 + 2X - 1$

```
LOAD
00: INP19    (X)
01: LDA19
02: MUL19    (X2)
03: STA18
04: LDA19    (X)
05: MUL10    (2X)
06: ADD18    (X2 + 2X)
07: SUB11    (X2 + 2X - 1)
08: OUT24
09: JMP00
10: 2
11: 1
12: END
```

CHALLENGE 3: NUMBER SEQUENCES

```
(a)          (b)          (c)          (d)
LOAD         LOAD         LOAD         LOAD
00: LDA04    00: LDA05    00: LDA05    00: OUT10
01: OUT24    01: OUT24    01: OUT24    01: LDA10
02: MUL05    02: MUL05    02: MUL06    02: ADD09
03: JMP01    03: ADD06    03: SUB07    03: STA09
04: 3        04: JMP01    04: JMP01    04: OUT09
05: 2        05: 2        05: 2        05: ADD10
06: END      06: 1        06: 3        06: STA10
             07: END      07: 1        07: OUT10
                          08: END      08: JMP02
                                       09: 0
                                       10: 1
                                       11: END
```

CHALLENGE 4: A DECISION MAKER

```
(a)                    (b)
LOAD                   LOAD
00: INP24              00: INP24
01: SKP05              01: SKP05
02: JMP05              02: JMP08
03: OUT07              03: LDA18
04: JMP00              04: STA22
05: OUT08              05: LDA17
06: JMP00              06: STA23
07: 1                  07: JMP00
08: 0                  08: LDA19
                       09: STA22
                       10: LDA17
                       11: STA23
                       12: JMP00
                       .
                       .
                       .
                       17: 010
                       18: 060
                       19: −060
                       20:
                       21: 12
```

(c)

```
LOAD
00: INP24 (X)
01: SUB08 (-7)
02: SKP04 (skip if <= 0)
03: JMP06 (jump if false)
04: OUT09 (true: 1)
05: JMP00
06: OUT10 (false: 0)
07: JMP00
08: 7
09: 1
10: 0
11: END
```

(d)

LOAD
00: INP18 (X)
01: LDA18 (X)
02: MUL18 (X^2)
03: SUB14 ($X^2 - 3$)
04: STA19 (save $X^2 - 3$)
05: LDA18 (X)
06: MUL15 (2X)
07: SUB19 ($(2X) - (X^2 - 3)$)
08: SKP03 (skip if = 0)
09: JMP12 (jump if false)
10: OUT16 (true)
11: JMP00
12: OUT17 (false)
13: JMP00
14: 3
15: 2
16: 1
17: 0
18: END

CHALLENGE 5: A SAWTOOTH

```
LOAD         LOAD21
00: LDA09    21: 12      (color)
01: STA23    22: 45      (turn)
02: LDA10    23: END
03: STA22
04: LDA9
05: STA23
06: LDA22
07: MUL11
08: JMP03
09: 10
10: 90
11: -1
12: END
```

CHALLENGE 6: A TWELVE-POINTED STAR

```
LOAD         LOAD21
00: LDA04    21: 12      (color)
01: STA23    22: 30      (turn)
02: MUL5     23: END
03: JMP1
04: 20
05: -1
06: END
```

APPENDIX II :

PROGRAM INSTRUCTIONS

Operation Code	*Mnemonic*	*Explanation*
1	LDAxx	Load the accumulator with the value in location xx.
2	STAxx	Store the value of the accumulator into location xx.
3	ADDxx	Add the value in location xx to the accumulator.
4	SUBxx	Subtract the value in location xx from the accumulator.
5	MULxx	Multiply the accumulator by the value in location xx.
6	DIVxx	Divide the accumulator by the value in location xx. This rounds off the answer down to the nearest one.
7	INPxx	Input to location xx.
8	OUTxx	Output from location xx.
9	JMPxx	Jump to location xx.
000	STP	Stop.
001	SKP01	Skip the next instruction if the accumulator is less than 0.
002	SKP02	Skip the next instruction if the accumulator is greater than 0.
003	SKP03	Skip the next instruction if the accumulator is 0.
004	SKP04	Skip the next instruction if the accumulator is either less than or equal to 0.
005	SKP05	Skip the next instruction if the accumulator is either greater than or equal to 0.
006	SKP06	Skip the next instruction if the accumulator is not 0.

APPENDIX III : *COMMANDS*

Command	*Explanation*
RUN	Run the program starting in location 00.
RUNxx	Run the program starting in location xx.
RUNSPEDx	Run the program using a speed of x. x can be a value from 0 to 9, with 0 being the slowest and 9 the fastest.
B	Break. The computer program will stop running.
CONT	Continue at preset speed.
CONSPEDx	Continue at speed x.
CONSTP	Continue, one step at a time.
NEW	Clears memory.
LOADxx	Get ready to load information beginning at location xx.

JOYSTICK CONTROL

The joystick can be used to perform the following functions:

BREAK	Press the red button to stop a program.
SPEED	After a RUN command, move the joystick forward to increase run speed, and backward to decrease it.
STEP	Push the joystick forward after a RUNSTEP command. This does the same thing as pressing the spacebar.

APPENDIX IV : *ERROR MESSAGES*

Error Letter	*Error Type*	*Explanation*
A	Overflow	Your calculation was larger than 999 or smaller than −999.
B	Undefined Value	You tried to do an arithmetic operation or an output operation using an empty memory location or accumulator.
C	Invalid Instruction	Either the mnemonic (or operation code) is missing or you have addressed a non-existent location.
D	Can't Continue	You stopped the computer in some way other than the B key or STP instruction.
E	Invalid Command	You used a command not in Simulated Computer's language.
F	Input Error	You typed letters when the computer expected numbers.
G	Missing Instruction	The computer is trying to get an instruction from an empty memory location.
H	Division By Zero	You tried to divide a number by zero. Sometimes you can use a SKP instruction to avoid this error.
I	Invalid Value	Color or sound value was too large or too small.

The Atari version of this program was created using valFORTH products of Valpar International, Tucson, AZ 85713, USA

Based on fig-FORTH, provided through the courtesy of Forth Interest Group, PO Box 1105, San Carlos, CA 94070.